

E: Remarks:

The present amendment has been prepared to comply with and be in accordance with the latest instructions on preparing amendments issued by the U.S. Patent And Trademark Office dated 1/31/03 which waives the provisions of 37 CFR 1.121(a), (b), (c) and (d).

The amendment is believed to be in accordance with the examiner's suggestions. The claims 15, 16 and 17 remain, with limitations of claim 4 and 14 included in Claim 15.

The examiner is thanked for his careful reading of the description. A substitute specification and drawings are submitted as discussed 4/20/2004 with the Examiner. No new matter has been included.

Before describing the changes to the description submitted as a substitute, the undersigned would like to point out that the original description as to the subject matter claimed was complete and correct. However, there were numbers used which were derived from other applications incorporated by reference, and those have been corrected here.

As to Figure 7, the picture is actually correct, but the explanation is a little involved. The difference is mainly because each L2 cache line can contain either 64B worth of data or 32B worth of instruction (that is 8 instructions, each 4B). If the line contains instruction (type bit = I), then using 32B allows room for keeping the "Hint Instructions".

**I-line:** If the line contains instructions, then it only has 32B worth of instructions. So to index the cache, we need 43:58 bits (drop the last 5-bits for 32B in each line). Bits 43:58 are 16 bits, and we need a total of 16-bits to index 0 through 65535 entries. The "Address of the first instruction" needs to be 0:42



bits. This is because, we drop the last 5-bits (from 0:63) since 32B in each line and we drop the next 16-bits to index 0 through 65535 entries. So, we only need to compare 0:42 bits from the directory for a hit.

**D-line:** If the line contains data, then it has 64B worth of data. Since it is double that of instruction, everything gets shifted to left by 1.

Accordingly, no change is necessary in the addressing used throughout the Figures as they are correct.

In the substitute specification, the following changes were generally made. The docket number of the referenced application became U.S. Patent 6598152 on July 22, 2003. That has been changed through.

Element 206 is shown on page 20, line 22, and page 30, line 23. Element 203 is on page 13, line 25 for Figure 2 and for page 16 line 1 for Figure 3. Element 311 was changed to 207 in Figure 3, as it is the element described as 207 instead of 311 on page 33 in this specification as shown in the drawing. 204 is shown in Figures 2 and 3 and described in page 10, line 12 and 13 and Page 16 line 2 for the BHT Table. The Figure 1 has been changed to Id\_bht op for identifying the op loaded into the field throughout, so the operation load was correct for the Id\_bht load operation for page 12, line 10, but the quotation mark was moved to make this clear. The element 211 was used twice because of the referenced application used that number for the branch execution unit with sequencing logic which here should be and is numbered 217A and 209, etc. The text in the second paragraph has been corrected to reference 209, 216, 217A in lieu of 214 and 211.



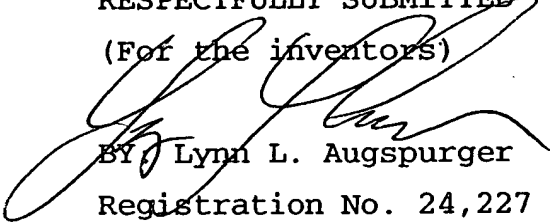
Page 36 has been correct to change Branch execution logic to 217A with outputs 316A and 316B.

These changes have been made throughout the description. No new matter has been introduced. The other changes suggested by the examiner in the action have been adopted.

It is respectfully submitted that the application should be in final condition for allowance which is respectfully requested.

RESPECTFULLY SUBMITTED

(For the inventors)

BY  Lynn L. Augspurger

Registration No. 24,227

Phone: 845-433-1174

Fax: 845-432-9601

Mailing Address:

IBM Intellectual Property Law

2455 South Road, P386

Poughkeepsie, NY 12601



**Branch Prediction Apparatus and Process for Restoring Replaced Branch  
History for Use in Future Branch Predictions for an Executing Program**

This invention deals with novel process and novel apparatus features which may be embodied in a single chip processor for significantly improving processor performance by enabling the restoration of branch predictions previously lost in a branch history table.

**TECHNICAL FIELD**

The present invention generally deals with increasing program execution performance by processor semiconductor logic chips. The improvement is obtained by uniquely preserving and enabling the reuse of branch history in a branch history table (BHT) for associated instructions replaced in an instruction cache (I-cache) of a processor. Prior branch prediction techniques using branch history tables have lost the branch history associated with instruction lines replaced in an instruction cache.

**RECEIVED**

AUG 03 2004

Technology Center 2100

**INCORPORATION BY REFERENCE:**

Incorporated by reference herein is the entire specification, including all disclosure and drawings, of application docket number PO9-96-134 having USPTO serial number 09/436264 filed on November 8, 1999 entitled "Increasing the Overall Prediction Accuracy for Multi-Cycle Branch Prediction Processes and Apparatus by Enabling Quick Recovery" invented by the inventor of the present application, now US Patent 6598152, granted July 22, 2003.

**BACKGROUND**

In prior art computer systems using branch history tables (BHTs), each BHT entry contains fields that predict the taken or not taken branch path for each branch instruction in an associated line of instructions in an instruction cache (I-cache). Each line of instructions contains N number of instruction locations, and each of the N instruction

1 locations may contain any type of instruction, e.g. a branch instruction or a non-branch  
2 instruction. There are N number of BHT fields in any BHT entry respectively associated  
3 with the N instruction locations in the associated I-cache line. Each BHT field may be  
4 comprised of one or more bits, and is sometimes referred to as a counter field. In the  
5 detailed example described herein, each BHT field comprises a single bit.

6 Any distribution of instruction types may exist in any I-cache line. Accordingly, a line of  
7 instructions within any I-cache entry may contain no branch instruction, or any  
8 combination of branch and non-branch instructions. For example, each I-cache entry may  
9 comprise an instruction line with 8 instruction locations, and each of these eight instruction

10 locations may contain an unconditional branch instruction, a conditional branch  
11 instruction, a non-branch instruction, or any other type of instruction. Thus, any  
12 distribution of instruction types may exist in any I-cache line. For example, the I-cache  
13 may have 32 K line entries. The I-cache index locates both an I-cache entry in the I-cache  
14 and an associated BHT entry in the BHT. Further, each BHT entry contains 8 BHT fields  
15 (e.g. bits) which are respectively associated with the 8 instruction locations in the associated  
16 I-cache entry. The only BHT bits in the BHT entry which are predictively effective are  
17 those associated with a branch instruction location, and the BHT bits associated with  
18 instruction locations containing non-branch instructions are ignored. For example, a BHT  
19 entry having a BHT bit set to a "1" state is predicting that a branch instruction in its  
20 associated location will be "taken", i.e. jump to a non-sequential instruction location on its  
21 next execution in the program. A "0" state for this BHT bit predicts its associated  
22 conditional branch instruction will be "not taken", i.e. go to the next sequential instruction  
23 location in the program. A BHT bit associated with an unconditional branch instruction is  
24 always set to the "1" state to indicate it is always "taken". The state of a BHT bit  
25 associated with a non-branch instruction is ignored, regardless of whether it has a "1" or  
26 "0" state.

27 In the prior art, a new line of instructions may be fetched from an L2 cache into an I-cache  
28 entry and replace a line of instructions previously stored in that I-cache entry. However,

1 the BHT entry associated with that I-cache entry is not replaced in the BHT when the  
2 instruction line is replaced in the I-cache entry. Whatever BHT prediction states exist in  
3 the BHT entry are assumed to be the predictions for the branch instruction(s) in the newly  
4 fetched line, even though the new line probably has branch instructions in different  
5 locations than the replaced I-cache line, and even though the existing BHT predictions may  
6 have been generated for other branch instructions in the program. Hence, the BHT  
7 predictions for a replaced line have a significant chance of providing wrong predictions for  
8 the branch instructions in the line.

9 When a BHT prediction selects the wrong branch path in the program, a sequence of  
10 incorrect instructions are selected and executed, because the selection of the wrong branch  
11 path is not immediately detected, but is detected many instruction execution cycles later.  
12 After detection, instruction results for these wrong instructions are destroyed, and the  
13 branch path is belatedly reset to the correct branch path from which the program  
14 execution continues, and the wrong BHT branch prediction is corrected in the BHT.  
15 Hence, wrong BHT predictions may cause significant time loss during program execution  
16 due to their selection of incorrect branch paths. This increase in the program execution  
17 time causes a corresponding reduction in the processing rate of executing programs. The  
18 resetting of wrong branch paths and the correction of BHT erroneous predictions is taught  
19 in the prior filed US application serial number 09/436264 (docket no. POU919990174). US  
20 Patent 6598152, granted July 22, 2003.

21 The statistical probability of BHT predictions being incorrect for a replaced line is  
22 variable. For example, if a newly fetched instruction line replaces a branch instruction  
23 with an unrelated branch instruction in the same I-cache location, the existing setting of its  
24 location associated BHT prediction is expected to have a 50 percent probability of being  
25 correct (and a 50 percent chance of being wrong). But if the new branch instruction in the  
26 newly fetched line replaces a non-branch instruction, and if this newly fetched instruction  
27 was the last branch instruction previously in that instruction location, its

1 location-associated BHT prediction has better than a 90 percent probability of being  
2 correct.

3 In the known prior art of BHT branch prediction techniques, the predictions in the branch  
4 history table were lost when associated branch instructions were replaced in the I-cache.  
5 The subject invention may be used with some of these prior BHT branch prediction  
6 systems to improve their BHT prediction rates.

## 7 SUMMARY OF THE INVENTION

---

8 This invention increases the speed at which a processor can execute a program by  
9 increasing the accuracy of its BHT branch predictions. This increases the processing  
10 speed of a program (even when there is no change in the instruction execution cycle time of  
11 the processor) by preventing the loss of previously-generated BHT predictions (which were  
12 lost in the prior art after replacement of associated branch instructions in the I-cache). For  
13 example, this invention may increase the BHT branch prediction accuracy for a branch  
14 instruction refetched to the same location in an I-cache entry - by increasing its probability  
15 of correctness from a potential 50 percent rate to in excess of a 90 percent rate. This is  
16 better than an 80 percent improvement in the prediction accuracy for branch instructions  
17 refetched in an I-cache, i.e. computed as  $(90-50)/50 = 80$ .

18 When an I-cache line of read-only instructions is replaced in to an I-cache, there is no  
19 castout of the replaced line because it has a copy available in the storage hierarchy for  
20 being refetched later into the I-cache. Also associated with that I-cache instruction line is a  
21 BHT entry which is not castout but may contain predictions that do not correctly predict  
22 the "taken or not taken" outcome of one or more branch instructions in the refetched line.

23 With this invention, when a line of instructions is replaced in the I-cache, the current state  
24 of its associated BHT entry is stored in a hint instruction in the I-cache. Later, the hint  
25 instruction is stored in the system storage hierarchy in association with a copy of the

1 replaced I-cache ~~replaced~~ instruction line. Also stored in that hint instruction are: a  
2 branch mask indicating the locations of any branch instructions within the replaced  
3 I-cache line.

4 In the detailed embodiment described herein, an associated hint instruction is generated  
5 and stored in the I-cache when the associated line is accessed therein. When the line is  
6 later replaced in the I-cache, its hint instruction is then stored in a second level cache in  
7 association with a copy of the replaced I-cache ~~replaced~~ instruction line. This invention  
8 may be used in hierarchy levels below the second level cache, such as a third level  
9 represented by the main memory of a system. When this invention is not extended to a  
10 third hierarchy level, the hint instruction is lost when its associated instruction line is  
11 replaced in the second level cache. Nevertheless, this invention is highly useful when it is  
12 only extended to the second level in the hierarchy, because line replacement in a large  
13 second level cache is rare. Extension to one or more additional storage levels is an  
14 economic tradeoff, whereby the cost of extension to a next hierarchy levels may be  
15 outweighed by the low frequency of instruction line's refetches at the lower hierarchy levels  
16 involving only a very small increase in program execution efficiency, such a fraction of 1  
17 percent. However, the subject invention comprehends the transfer and storage of hint  
18 instructions to one or more storage levels beyond the second level cache in the system  
19 storage hierarchy.

20  
21 In more detail, during an I-cache hit a hint instruction is generated and stored with its  
22 instruction line in a row of the I-cache to associate the hint instruction and the I-cache  
23 instruction line. When an I-cache miss occurs for the instruction line, the hint instruction  
24 is transferred from the I-cache to a row in the L2 cache containing the L2 copy of the  
25 associated instruction line. Then the I-cache line and its hint instruction are replaced by  
26 another instruction line and hint instruction copied from a row in the L2 cache located by  
27 the current instruction address (in IFAR). The replacing hint instruction will be a  
28 developed (generated) hint instruction if its L2 copy was previously used during the  
29 current execution of its program, i.e. the line is being fetched again (i.e. refetched) into the



1 I-cache - then its associated hint instruction is fetched and used to restore predictions in the  
2 current BHT entry for branch instructions in the refetched line. This BHT entry  
3 restoration process does not affect its BHT bits corresponding to non-branch instructions  
4 in the refetched line. Thus, the restoration can only affect BHT predictability for branch  
5 instructions in the newly fetched instruction line and does not affect the predictability of  
6 BHT bits associated with non-branch instructions in the associated instruction line. A  
7 "branch mask" in the hint instruction aids in the restoration by indicating the locations of  
8 any branch instructions in its associated instruction line.

9 Thus, the number of restored bit positions in a BHT entry is dependent on the number of  
10 branch instructions in the associated replaced line, and the branch instruction locations in  
11 the line are indicated by the branch mask in the hint instruction. If all instruction  
12 positions in a replace line contain branch instructions, all predictions in the associated  
13 BHT entry may be restored by this invention. But if less than all predictions in the  
14 associated BHT entry contain branch instructions, less than all BHT fields in the associated  
15 BHT entry are restored by this invention. Most instruction lines have less than all of its  
16 locations containing branch instructions, and some instruction lines have no branch  
17 instructions.

18 In the described embodiment, each hint instruction contains an operation code (op code) to  
19 identify a developed hint instruction, which contains a BHT index (bht\_index) that locates  
20 the associated BHT entry in the BHT, a branch mask (branch\_mask), and a BHT entry  
21 (bht\_bits) which stores a copy of the BHT entry having the BHT states existing when its  
22 associated instruction line was replaced in the I-cache. The branch mask has a "1" mask  
23 bit at each BHT field position associated with a branch instruction position in the  
24 associated instruction line. A "0" mask bit is provided at each branch mask position  
25 corresponding to a non-branch instruction position in the associated instruction line. In a  
26 restored BHT entry, the only changeable BHT positions correspond to the "1" positions in  
27 the branch mask. During the restoration process, each BHT field position in the BHT  
28 entry located at a corresponding "1" state mask-bit position is set to the state of the

1 corresponding prediction position in the BHT field (bht\_bits) stored within the same hint  
2 instruction. In the BHT entry, no change is made to each BHT field position located by a  
3 “0” state mask-bit position.

4 Consequently, this invention allows the “0” mask bit positions in a restored BHT entry to  
5 represent predictions made for branch instruction(s) in different instruction lines that may  
6 later be refetched into the associated I-cache entry, as long as those branch instruction(s)  
7 are at non-branch locations in the currently replaced instruction line.

8 Accordingly, the process of this invention increases BHT prediction accuracy by enabling  
9 each BHT entry for a refetched instruction line to restore only the BHT predictions for the  
10 branch instruction positions in the refetched line. The avoidance of changing BHT  
11 predictions at non-branch instruction positions in a restored BHT entry has the useful  
12 benefit of allowing the non-branch BHT positions to retain predictions previously made for  
13 another instruction line that may in the future be refetched. This allows a restored BHT  
14 entry to retain predictions for multiple different instruction lines when such predictions  
15 are located at BHT positions which will not be used by any instruction in the currently  
16 associated line.

17 Novel apparatus is described in the detailed embodiment to support this inventive process  
18 by modifying both the I-cache and the second-level cache to receive and store hint  
19 instructions in association with instruction lines stored therein. This is done in both the  
20 first level I-cache and the second level cache by structuring each row in each cache to store  
21 both an instruction line and an associated hint instruction. The hint instruction location  
22 in each row is initialized by storing therein a “no operation” (NOP) type of hint instruction.  
23 This may be done by using a NOP code in the operation code field of a hint instruction and  
24 ignoring all other fields in the NOP instruction when it is detected as a NOP. The first time  
25 during a program execution an instruction line is fetched into the I-cache from the L2  
26 cache in response to a current cache miss, the accessed L2 cache row will have been  
27 initialized with a NOP hint instruction, and this instruction line and its NOP are copied

1 into the I-cache row having the current cache miss. The NOP may contain all "0" states in  
2 its "branch\_mask" and "bht bits" fields to prevent any restoration in the associated BHT  
3 entry at this time. However, if this instruction line thereafter has an I-cache hit, a real hint  
4 instruction (in the form described above) is generated and stored over the NOP hint  
5 instruction in the associated I-cache row. Later when this I-cache line has a miss, this real  
6 hint instruction is copied from the I-cache row to overlay the corresponding NOP hint  
7 instruction in the L2 cache row containing a copy of the instruction line having the cache  
8 miss. Then the line and hint instruction are replaced in that I-cache entry. Then during  
9 the continuing execution of the program, this L2 stored hint instruction is available to  
10 restore its associated BHT entry when and if its associated instruction line is refetched  
11 from the L2 cache into the I-cache. The restored BHT entry fields then have the benefit of  
12 using the latest prediction for their associated instructions, thus having a greater chance of  
13 representing a correct BHT prediction.

14 Hence, it is the primary object of this invention to reduce the occurrence of wrong BHT  
15 predictions for a program by the restoration of BHT predictions lost in an I-cache by  
16 replacement of instruction lines therein without affecting associated BHT predictions  
17 which cannot be currently used. The invention increases processor execution speed by  
18 expanding the amount of branch history available to an executing program beyond the  
19 prediction capacity of the BHT, and this invention makes the replaced branch history  
20 quickly available from another level of the system storage hierarchy for later use during  
21 execution of a program.

22 The restoration process of this invention may overlap the normal operation of standard  
23 cache operations so that little or no processor execution time need be lost when this  
24 invention is used.

25 This invention discloses and claims novel "hint instruction" micro-control processes and  
26 apparatus which can operate in parallel with the normal program instruction processing  
27 controls of a processor to enable BHT predictions for replaced branch history to be stored

1 in a usable form at another level in a storage hierarchy from which it can be quickly  
2 retrieved and used by an executing program. The micro-controls disclosed and claimed as  
3 part of this invention are preferably embedded in, and part of, the same semiconductor  
4 chip that contains the processor executing the program. Novel "hint instructions" are  
5 generated and used by the novel processes disclosed and claimed herein in these the  
6 micro-controls.

7 The hint instructions may operate transparent to a program executed with conventional  
8 program instructions, while hint instructions are being concurrently generated and  
9 executed by the "hint processing" micro-controls in the same chip as the processor  
10 executing the program.

11 Both an instruction line and an associated hint instruction may be stored in the same row  
12 of an L1 cache and an L2 cache. The L1 and/or L2 cache structure may be designed using  
13 separate subarrays, one subarray for storing the program instruction lines (i.e. in a  
14 "instruction cache" subarray), and the other subarray for storing the associated hint  
15 instructions (i.e. in a "hint instruction" subarray). This modified structure may have the  
16 advantage of enabling each subarray to have a bit width that is a power of 2, which is a  
17 design preference with some cache designers. Then the line index for selecting a line in the  
18 cache subarray would also be used to select the associated hint instruction in the "hint  
19 instruction" subarray. Part of the same IFAR address selects the BHT entry in a separate  
20 BHT subarray .

21 In the detailed embodiment described herein, the term "hint instruction cache (HIC)" is  
22 generally used to identify a novel I-cache in which each row stores both an instruction line  
23 and its associated hint instruction.

24 Thus, this invention provides a novel hint instruction having novel controls using novel  
25 hardware and novel processes, which enable the saving and fast utilization of branch  
26 history for instructions replaced in an I-cache - to store their branch history elsewhere in

1 the storage hierarchy, which if lost would require the inefficient process of resetting more  
2 wrongly-selected branch paths and belatedly redeveloped BHT predictions to replace the  
3 lost BHT predictions.

#### 4 BRIEF DESCRIPTION OF THE DRAWINGS

5 FIGURE 1 illustrates the hint instruction form used in the described embodiment of the  
6 subject invention.

7 FIGURE 2 shows an overview of instruction execution controls in a processor having the  
8 novel hint instructions and processes shown in the other FIGUREs for the detailed  
9 embodiment.

10 FIGURE 2A shows the hint processor represented in FIGURE 2. FIGURE 2B represents  
11 the hardware logic of the “new BHT creation logic” circuits in FIGURE 2A.

12 FIGURE 3 is a modified view of the hint instruction controls represented in Figure 2.

13 FIGURE 4 represents the branch information queue (BIQ), and the form of its queue  
14 entries shown in block form in FIGURE 3.

15 FIGURE 5 represents a branch history table (BHT) associated with the Hint Instruction  
16 Cache IL1 seen in the block diagram of FIGURE 2.

17 FIGURE 6 shows the general form of the novel hint instruction cache (IL1) and its  
18 instruction cache Directory (IL1 Dir) shown in FIGUREs 2 and 3.

19 FIGURE 7 shows the general form of a L2 Cache Directory and its novel associated L2  
20 Cache shown in block diagram form in FIGURE 2.

**FIGURES 8, 9 and 10 are flow diagrams that include hint instruction processes according to the subject invention which operate during the execution of program instructions for extending branch-history predictive operations in a branch history table (BHT).**

**FIGURE 11 shows a flow diagram of an Instruction Decode and Dispatch subprocess used in FIGURE 9.**

**FIGURE 12 shows a flow diagram of an Instruction Issue and Instruction Execution subprocess used in FIGURE 9.**

**FIGURE 13 shows a flow diagram of the subprocess performed by the hint processor shown in FIGURE 2A.**

## **DESCRIPTION OF THE DETAILED EMBODIMENT**

**The detailed embodiment described herein has novel processor hardware shown in block form in FIGUREs 2, 2A, 3, 4, 5, 6 and 7, which may be structured on a processor chip, and FIGURES 8, 9, 10, 11, 12 and 13 represent the detailed novel process and novel subprocesses performed by the illustrated hardware.**

**FIGURE 2 includes a novel hint instruction cache (IL1) 201 and a novel L2 cache 212, each capable of containing a multiplicity of novel hint instructions, and conventional program instructions. Program instructions are fetched from the L2 cache 212 into the instruction cache (IL1) 201 for execution by the program currently being executed in the processor. The hint instructions in L2 cache 212 and in IL1 201 are each located in a respective row containing a line of instructions. In each cache, an association is obtained between an instruction line and a hint instruction by their being placed in the same cache row.**

**Either real addresses, or virtual addresses translated in the conventional manner by the processor, may be used by the executing program to address program instructions and**

1 data in a main storage of the processor system, and in each of the caches through their  
2 respective cache directories in the conventional manner. Any size virtual addresses may be  
3 used, such as 64 bit or 32 bit addresses.

4 FIGURE 1 shows the form of each hint instruction 100 and NOP hint instruction 109  
5 stored in caches 201 and 212 in FIGURE 2. The hint instructions are each shown as a 32  
6 bit instruction. The hint instructions may operate within the processor in a manner  
7 transparent to the executing program.

8 NOP (non-operational) instruction 109 is used for initializing the space to be occupied by a  
9 hint instruction 100, and the NOP format contains only the NOP code in the first 5 bits of  
10 the instruction and its remaining bits are unused. Hint instruction 100 has a load BHT  
11 operation code in its five bit positions 0-4 labeled "~~ld\_bht op~~" "ld\_bht op". The NOP  
12 instruction type is used in the described embodiment to initialize storage space which later  
13 may be filled with the "ld\_bht op" load hint instructions. In this embodiment, the load  
14 BHT hint instruction and the NOP instruction are each 4 bytes long, i.e. 32 bits. The length  
15 of each field in these instructions are indicated by dimension arrows in each of the  
16 instructions in FIGURE 1, and each dimension arrow is labeled with a centered bit number  
17 to indicate the bit length of its respective field. Thus, instruction 100 includes the five bit  
18 "ld\_bht op" field, an eleven bit "bht\_index" field, an eight bit "branch mask" field  
19 (br\_mask), and an eight bit "bht\_bits" field. As previously stated, the "ld\_bht op" field is  
20 the operation code of instruction 100. The bits in the "bht\_index" field provide the 48:58 9  
21 index to locate and associate an IL1 cache entry (containing an instruction line), its IL1  
22 directory entry, and their associated BHT entry. The "branch mask" field contains 8 bits,  
23 and each branch mask bit corresponds to a respective one of the 8 instruction locations in  
24 the associated instruction line. A mask bit is set to the "1" state to indicate when its  
25 respective instruction location contains a branch instruction, and is set to the "0" state to  
26 indicate when its respective IL1 instruction location does not contain a branch instruction.  
27 The "bht\_bits" field stores the content of a BHT entry located at the "bht\_index" in the  
28 BHT for the BHT entry associated with an instruction line being replace in the IL1 cache.

1 Each hint instruction is generated and stored in the hint instruction location identified by  
2 the current IFAR address, when the associated instruction line in IL1 cache 201 is being  
3 accessed with a cache hit.

4 A hint instruction is executed when its associated instruction line has a cache miss in the  
5 IL1. Then, the associated hint instruction is used to change the associated BHT entry if the  
6 associated instruction line has any branch instruction(s). The change in the associated  
7 BHT entry is only at a BHT bit located at a “branch mask” bit position having a “1” state  
8 (indicating the corresponding instruction is a branch instruction), if the “branch mask”  
9 has any “1” bit. Then, only the “1” mask bit position(s) are located in the current BHT  
10 entry where they are set to the “1” or “0” bit state of the corresponding bit position in the  
11 “bht\_bits” field of the hint instruction, i.e. only at the “1” mask bit position(s) in the BHT  
12 entry. The “0” mask bit locations in the associated BHT entry are not affected by the  
13 process of executing the associated hint instruction.

14 During an IL1 cache miss, the associated hint instruction stored in the IL1 cache 201 is  
15 copied to the L2 cache immediately before the associated instruction line in the IL1 cache  
16 201 is overlayed in the IL1 cache by a new instruction line fetched from the L2 cache. The  
17 L2 location for this hint instruction is generated from the content of the associated IL1  
18 directory entry, i.e. from a “address of the first instruction” field that indicates the address  
19 of the first instruction to be executed in the associated instruction line.

20 Generally, a NOP instruction marks an entry location in the L2 cache which does not  
21 contain any IL1 replaced entry. That is, a NOP indicates an L2 entry which may contain a  
22 copy of an instruction line that have not been replaced in IL1 201, although it may have  
23 been copied into the IL1 cache where it currently exists. A NOP instruction is overlayed by  
24 a newly generated “Ld\_bht” instruction when its corresponding IL1 location is first used in  
25 the IL1 cache 201.



1 An IL1 index 48:58 is used to locate a row of IL1 instructions in IL1 201 and its  
2 corresponding IL1 directory entry in directory entry 202. The IL1 index is obtained from  
3 the eleven address bit positions 48 through 58 (i.e. 48:58) in IFAR 203 in FIGURE 2. The  
4 rows in the IL1 cache is shown divided into two sections 201A and 201B (FIGURE 3) which  
5 respectively contain the instruction lines and the hint instructions. However, these sections  
6 may instead be obtained by using separate hardware subarrays which are accessed with  
7 the same index 48:58. The value of the 11 bits obtained from the sequence of address bit  
8 positions 48:58 locates one of 2048 rows in IL1 201 and also locates the corresponding  
9 row in IL1 directory 202 to associate the IFAR address with these two selected rows.

---

10 The IL1 index bits 48:58 of the IFAR address are also used to select a BHT entry in a BHT  
11 204, shown in FIGURES 2 and 3. Thus, IFAR bits 48:58 associate a BHT entry in BHT 202  
12 with an instruction line and a hint instruction in IL1 201 and its corresponding directory  
13 entry in the IL1 directory 202.

14 IL directory 202 is a conventional and contains a "valid bit" field and a 48 bit "address of  
15 the first instruction" (i.e. first instruction address) field. A valid state in the valid bit field  
16 indicates that the associated IL1 row (in IL1 201) contains an instruction line, and that the  
17 "first instruction address" field locates the first program instruction to be selected for  
18 execution in that instruction line. An invalid state of the valid bit indicates the contents of  
19 the corresponding IL1 row are invalid and should not be used.

20 In this embodiment, each IL1 row contains space for eight 32 bit program instructions and  
21 one 32 bit hint instruction shown at the end of each row. Hence, each program instruction  
22 and each hint instruction has an instruction length of 4 bytes (i.e. 32 bits), in the respective  
23 columns 201A and 201B of IL1 201. In the IL1 directory row, each "first instruction  
24 address" field contains 48 bits which locate the first program instruction to be accessed in  
25 the corresponding row in IL1 201.

1    **The first instruction address" field in the IFAR selected IL1 directory row is used only if**  
2    **the content of the "first instruction address" field in that row matches the current address**  
3    **bits 0:47 in IFAR 203. When a compare-equal occurs between IFAR bits 0:47 and the**  
4    **"first instruction address" field in the accessed IL1 directory row, the addressed first**  
5    **instruction is allowed to be used in the associated IL1 row.**

6    **In FIGURE 2, the BHT 204 operates with IL1 201 to provide a prediction of whether the**  
7    **branch instructions stored in IL1 201 are to be "taken" or "not taken" in the program**  
8    **being executed. Generally, a "taken" branch instruction indicates the instruction path is to**  
9    **go to the address indicated by that instruction, and a "not taken" branch instruction**  
10    **indicates the instruction path is to continue with next sequential instruction in the**  
11    **program.**

12    **Each BHT entry in this embodiment contains eight 1-bit prediction fields. The sequence of**  
13    **the eight 1-bit prediction fields in any BHT row respectively provide a prediction of the**  
14    **"taken" or "not taken" state for each branch instruction at the corresponding position in**  
15    **the line. A BHT bit is ignored when its corresponding program instruction is not a**  
16    **conditional branch instruction. Thus, the only meaningful prediction bit(s) in any BHT**  
17    **row are those that correspond to a conditional branch instruction in the associated IL1**  
18    **row. The 0 state of a BHT prediction bit indicates it is predicting the "not taken" state for**  
19    **any corresponding conditional branch instruction, and the 1 state of a prediction bit**  
20    **indicate it is predicting the "taken" state for any corresponding conditional branch**  
21    **instruction.**

22    **FIGURE 3 shows in more detail parts of FIGURE 2 and shows it from another perspective**  
23    **to aid in the teaching the operation of the detailed embodiment of this specification. Thus,**  
24    **FIGURE 3 shows IL1 201 in more detail as having a section 201A containing program**  
25    **instructions and a section 201B containing hint instructions. That is, each row of**  
26    **instructions in IL1 201 has its leftmost part in section 201A for containing program**  
27    **instructions, and its rightmost part in section 201B for containing a hint instruction at the**

1 end of each row. Section 201A operates as an Instruction Cache (I-cache) of the type  
2 described in the incorporated US Patent 6598152, granted July 22, 2003 patent application  
3 ~~(attorney docket number POU919990174)~~ having USPTO serial number 09/436264 for  
4 increasing the overall prediction accuracy for multi-cycle branch prediction processes and  
5 apparatus for enabling quick recovery in generating new prediction for the BHT.

6 As previously mentioned, the address in IFAR 203 selects a row of program instructions in  
7 IL1 201 and an associated BHT row of prediction fields in BHT 204. FIGURE 3 illustrates  
8 the IFAR selected BHT row (i.e. also herein called the current BHT entry) being outputted  
9 to an "eight prediction" register 308, and the IFAR selected IL1 row (i.e. a group of 8  
10 program instruction fields) being outputted to an "eight program instructions" register  
11 309. Each branch instruction field in register 309 has an associated branch prediction field  
12 at a corresponding location in register 308. The associated branch prediction field is only  
13 used if the corresponding branch instruction field contains a conditional branch  
14 instruction. Hence, the associated branch prediction field is not used if the corresponding  
15 instruction field contains a non-branch instruction.

16 The "branch taken / not taken" state of each branch prediction bit in register 308 (when  
17 associated with a corresponding conditional branch instruction in register 309) is generally  
18 determined by the most-recent execution of that instruction in the current program. The  
19 correctness of this branch prediction is checked by branch execution logic 214  
20 (209,216,217A) after the IFAR selection of the corresponding branch instruction in the  
21 program. Whenever the check by branch execution logic 214 (209,216,217A) finds a BHT  
22 prediction is correct, the last predicted execution path continues to be followed in the  
23 program without interruption. But when execution logic 214 (209,216,217A) finds a BHT  
24 bit prediction is wrong, the wrong path has been followed in the program, the correct path  
25 must be found and followed, and the execution results of the wrong path are discarded.  
26 Thus, when logic 214 (209,216,217A) finds the currently BHT prediction bit is wrong, the  
27 correct setting for that BHT bit also is determined, and the state of that BHT bit is changed  
28 to its correct state. The target address of the executed branch instruction is then known

1 and is determinative of the location in the program execution from which the incorrect  
2 path began and is the beginning of the correct new execution path in the program.

3 This manner of operation for re-setting the execution path when a wrong prediction is  
4 detected by logic 211 (209,216,217A) is described and claimed in the incorporated US  
5 Patent 6598152, granted July 22, 2003 ~~specification (POU919990174 having USPTO serial~~  
6 ~~number 09/~~ ~~filed on November 8, 1999)~~. In more detail regarding this  
7 incorporated specification, whenever a new row of instructions was fetched into its  
8 instruction cache (I-cache) from a storage hierarchy, these newly fetched instructions  
9 overlay and replace any and all instructions previously stored in that I-cache row. In this  
10 prior system, the BHT entry associated with that I-cache row are not replaced in the BHT  
11 when the associated IL1 row received the newly fetched instruction line. Thus, whatever  
12 pre-existing prediction states exist in the BHT entry (determined by execution of the  
13 replaced instructions in the row no longer in the I-cache) are then used as the branch  
14 predictions for the new unrelated branch instruction(s) newly fetched I-cache row and  
15 overlaying the corresponding that were used to generate those BHT bit states. When any  
16 BHT prediction bit is used and then later found incorrect by execution logic 211  
17 (209,216,217A), the bit is belatedly rewritten in its BHT location to correct it. The penalty  
18 for incorrectness of any BHT bit prediction is the loss of all execution results obtain from  
19 instructions executed in the wrong execution path and the time taken for such wrong  
20 executions. Hence for each BHT bit correction, many unneeded instructions may have  
21 been selected and executed, wasting many instruction selection and execution cycles which  
22 detract from the required program execution and decrease the program execution speed.

23 The rate of incorrect predictions is decreased by this invention enabling recent branch  
24 history lost in the prior art operation (when instructions are replaced in an instruction  
25 cache) to be retained in hint instructions and reused. The subject invention increases the  
26 likelihood of the associated BHT prediction field being correct for a I-cache row of  
27 instructions re-fetched into the instruction cache - by enabling the saving and reuse of the

1 prediction fields associated with overlaid rows of instructions in an I-cache whenever that  
2 row of instructions is refetched during later execution of a program..

3 The top of the storage hierarchy in FIGURE 2 is herein called "level 1" in the storage  
4 hierarchy of the system and contains the instruction cache IL1 201 and a data cache  
5 (D-cache) 221. They respectively provide the instructions and data to the central processor  
6 for execution by the current program. The next level in this hierarchy is called "level 2"  
7 which provides the instruction lines and data to the level 1 caches, and herein is provided  
8 by the L2 cache 212 which contains both instructions and data. It provides instruction  
9 lines to IL1 201 and lines of data to D-cache 221 in response to demands (misses) by the  
10 level 1 caches. L2 cache 212 obtains its instructions and data from the main memory of the  
11 computer system in the conventional manner of storage hierarchies.

12 The L2 cache of this invention has the unique structure for containing hint instructions  
13 which is not found in the prior art.

14 In IL1 201 (see FIGURE 6) and in the L2 cache with hint extensions (see FIGURE 7), the  
15 program instructions and the hint instruction are stored in predetermined locations in each  
16 of the cache entries to distinguish the program instructions from the hint instruction stored  
17 in the same cache entry. Thus the left part of each IL1 and L2 cache row contains space  
18 for storing a line of program instruction, and the right part of the row contains space for  
19 storing a hint instruction or a NOP instruction. The hint instruction locations in both the  
20 IL1 and L2 caches are initialized to contain NOP instructions, which are overlaid whenever  
21 a hint instruction is to be stored into the cache entry.

22 Thus initially during program execution, NOP instructions exist in the hint instruction  
23 locations in the IL1 and L2 caches. When an initial miss occurs for a program instruction  
24 line in both the IL1 cache entry, and in the L2 cache, the line of program instructions  
25 (containing the requested instruction) is fetched from system main storage into that line  
26 location in the L2 cache, and also into the IL1 cache entry. Later during the program

1 execution, the space occupied by this IL1 cache entry may be needed for a new line of  
2 program instructions which maps to this same IL1 cache entry during execution of the  
3 program. Before the new line is allowed to be stored into this IL1 cache entry, the existing  
4 line in the IL1 cache entry is replaced in the IL1 cache and a hint instruction is stored into  
5 the L2 cache entry having the copy of the replaced instruction line with the hint instruction  
6 generated for the BHT entry of the replaced instruction line.

7 In FIGURE 2A, each hint instruction is generated in the detailed embodiment by “hint  
8 instruction generation” circuits in the hint processor when required during the program  
9 instruction executions. The detailed embodiment uses “hint instruction execution” circuits  
10 in the hint processor to execute each hint instruction when required during the program  
11 instruction executions. Alternatively to having separate hint processor hardware circuits,  
12 the same hint processor generation and execution functions may be provided by having  
13 these functions micro-coded as subprocesses in the central processor executing the  
14 program.

15 The general operation of the IL1 cache with concurrent hint instruction suboperations is  
16 done in FIGURES 2 and 3 as follows: When the central processor in FIGURE 2 needs to  
17 select an instruction, the IL1 cache row is selected by IFAR bits 48:58 (i.e. the current  
18 instruction address is in IFAR). If that row contains the IFAR addressed instruction, it is  
19 accessed to provide an IL1 hit. If that instruction is not found therein, an IL1 miss occurs.

20  
21 An IL1 cache miss may occur under two different conditions: (1) The valid bit in the  
22 associated IL1 directory entry may be indicating the invalid state, which will cause a cache  
23 miss. (2) When that valid bit is indicating a valid state, a cache miss occurs if the current  
24 IFAR address bits 0-47 do not match the current address in the “address of the first  
25 instruction” field in the associated IL1 directory entry.

26 If an IL1 cache miss occurs for reason (1), i.e. because the directory valid bit indicates the  
27 invalid state, no valid instruction line exists in this IL1 cache entry and a new instruction

1 line may immediately be fetched from the L2 cache and copied into that IL1 cache row.  
2 The hint instruction associated with the L2 cache copy of the line is copied into the hint  
3 instruction location in the same IL1 row. The form of the copied hint instruction is the  
4 form found in the copied L2 cache row, which is either 100 or 109 in FIGURE 1.

5 However, if an IL1 cache miss occurs because of reason (2), i.e. the directory valid bit  
6 indicates the valid state when the IL1 directory entry's "address of the first instruction"  
7 field does not match the current IFAR address bits 0-47, a valid instruction line exists in  
8 the IL1 cache row with an associated hint instruction, and the hint instruction must be  
9 castout to the L2 cache row containing a copy of the IL1 instruction line before it is  
10 overlaid by a hint instruction associated with the instruction line being fetched. This L2  
11 cache row is located by using the "address of the first instruction" field in the associated  
12 IL1 directory entry.

13 It will be recalled that current programs are comprised of read-only code which is not  
14 changed during execution of a program. Therefore the read-only program instructions in  
15 an existing line in IL1 201 do not need any castout operation (as is required for data  
16 changed by the program instructions in D-cache 221). Therefore, no castout is required for  
17 an IL1 line of program instructions about to be overlaid, since the line can be later  
18 obtained from a corresponding line in some level of the system storage hierarchy. A line of  
19 program instructions in an IL1 cache entry usually has a copy in a corresponding L2 cache  
20 entry, and the corresponding L2 cache entry may have copies at other levels of the storage  
21 hierarchy.

22 Hint instructions are generated and written into the IL1 and L2 cache rows by the hint  
23 instruction generation process when an instruction hit occurs in IL1. A hint instruction  
24 100 is generated and written into the hint instruction location in an associated IL1 row by  
25 the hint processor 206. This hint instruction generation process uses the current IFAR  
26 address and the associated line of program instructions to generate the fields in each hint  
27 instruction.

1 When a valid instruction line exists in the IL1 row having a miss, its associated hint  
2 instruction is executed by the hint processor 206 concurrent with its castout to its L2 cache  
3 row and while the newly fetched line of instructions is being written in the IL1 cache entry  
4 to overlay the associated line. The newly fetched hint instruction (from the IFAR  
5 addressed L2 cache row) is written into the hint instruction location, overlaying the  
6 executed hint instruction in that location.

7 It is to be noted that on any IL1 cache miss, the replacement new line of instructions is  
8 obtained from a different L2 cache entry than the L2 cache entry containing a copy of the  
9 replaced IL1 cache line causing the miss. It is further to be noted that branch instruction  
10 distribution in the replacement line may be independent of the branch instruction  
11 distribution in the replaced line. This has implications in the content of their BHT  
12 prediction values by indicating that each has a BHT content independent of the other.

13 The L2 cache is generally much larger than the IL1 cache and has many time more entries  
14 than the IL1 cache. The L2/IL1 entry ratio is preferably a power of two. In the described  
15 embodiment a ratio of 32 ( $32=2^{**5}$ ) is used. A small L2 cache may have twice the number  
16 of entries of the IL1 cache. An expected common occurrence during the IL1 cache misses  
17 for many IL1 cache entries is to have a replacement sequence for an IL1 cache entry which  
18 alternates between two different L2 cache instruction lines, which are respectively  
19 associated with two different hint instructions. These two instruction lines may have one or  
20 more branch instructions at different locations, and/or one or more branch instructions at  
21 the same location within their respective lines. This different branch instruction  
22 distribution characteristic can affect their respective BHT values during the operation of  
23 this invention.

24 A hint instruction stored in the L2 cache enables the current program to refetch the  
25 associated line from the L2 cache and restore the associated BHT prediction bits to the  
26 most recent prediction state(s) for any branch instructions in the line without disturbing



1 the prediction states for any non-branch bit positions in the BHT entry. The implication of  
2 this is that the undisturbed states of the non-branch positions may continue to represent  
3 the latest predictions for any branch instruction(s) in an alternate instruction line when it  
4 is not stored in the IL1 row to which it maps. These BHT bit predictions for the  
5 non-branch positions have the advantage of not needing to be regenerated when the  
6 alternate line for which they were generated is later refetched into that IL1 row; whereby if  
7 their states were disturbed it would increase the chance of selecting one or more wrong  
8 execution paths when the alternate line is again written in that IL1 cache row.

9 In this manner, the BHT prediction bit states for branch mask positions in the hint  
10 instructions stored in the L2 cache provide "hints" of the most recently used  
11 "taken/non-taken" branch state of each conditional branch instruction in their associated  
12 lines of instructions, whereby the mask indicated positions have a greater than 90% chance  
13 of providing correct predictions, instead of merely the 50% chance of providing a correct  
14 prediction if they were represented by the BHT values for the last instruction line in that  
15 IL1 cache entry.

16 In this manner, the hint instructions can restore the BHT bits for the branches in refetched  
17 IL1 lines to the prediction states most likely to correctly predict the branch outcomes.  
18 Thus the result of using the hint instructions of this application is to save processor  
19 execution time that would otherwise be lost in executing unnecessary instructions in  
20 wrongly selected execution paths in a program due to using incorrect BHT bit states for  
21 replaced IL1 lines.

22 FIGURE 7 shows the form of the L2 cache 212 and its directory 211 in the described  
23 embodiment. IFAR address bits 43 through 58 (43:58) are used as an index to locate and  
24 to associated a L2 cache entry and its corresponding L2 directory entry. Each L2 directory  
25 entry contains a "type bit" for indicating whether the addressed cache row contains an  
26 instruction line (I) or a data line (D). For example, type "1" may indicate a line of  
27 instructions, and type "0" may indicate a line of data words. Each L2 directory entry also

1 contains the "address of the first instruction" in its associated line and a valid bit to  
2 indicate if the addressed line is valid.

3 The IL1 cache and the L2 cache used in the detailed flow diagrams in FIGURES 8 - 13 are  
4 shown in FIGURES 6 and 7, in which the IL1 cache is a dedicated instruction cache which  
5 only contains instructions, which in this specification can have two types of instructions  
6 stored therein: "program instructions" and novel "hint instructions". There also is a IL1  
7 data cache 221 which contains the data accessed by the operands in the instructions  
8 executed from the IL1 instruction cache. This invention may also use a unified IL1 cache  
9 (not shown) containing both instructions and data.

10 In the detailed embodiment, a unified L2 cache is shown and used; it is a unified cache  
11 because it contains both instructions and data. Data cache operations are not used and are  
12 not needed in explaining this invention being claimed in this specification. In the  
13 corresponding L2 directory entry an "I" or "D" indication in a predetermined field  
14 indicates whether the associated line contains instructions or data, when the valid bit is set  
15 to the valid state in that L2 directory entry.

16 Each L1 and L2 cache row has space for a line of instructions and space for an associated  
17 hint instruction; the hint instruction space is in a predetermined location in each row,  
18 which may be anywhere in its row but is shown herein at the end of its line of instructions.  
19

20 Other tag bits (not shown) may also be included in each directory entry, for example, an L2  
21 directory entry containing a "D" indicator may also contain a "change bit" (not shown) to  
22 indicate if the data in the corresponding L2 cache entry has been changed since it was  
23 received by that L2 cache entry, whereby a castout of the contained data line need only be  
24 done if the data is indicated as having been changed. An "I" indication in a L2 directory  
25 entry does not need any "change bit" because the program instructions are not changeable  
26 in any cache entry.

1 Program instructions and data are fetched from the system storage hierarchy to the L2  
2 cache entries in response to an L2 cache miss. Program instructions are fetched from L2  
3 cache entries to IL1 cache entries in response to an IL1 cache miss.

4 However, only changed data in the L2 cache is castout to the system storage hierarchy  
5 when the data is to be replaced in an L2 cache entry. No castout is done for program  
6 instructions, because all program instructions are presumed to be readonly and  
7 unchangeable in both the IL1 and L2 caches.

8 A line of program instructions may remain valid in the L2 cache entry as long as its L2  
9 cache space is not needed for other program instructions. The mask-located prediction bits  
10 in any BHT field in the L2 hint instruction remain usable as long as its associated line of  
11 program instructions is valid in the L2 cache. A BHT entry may later be restored by a hint  
12 instruction when the associated line of program instructions is later retrieved from a valid  
13 L2 cache entry having a hint instruction. The restored BHT prediction bits in a BHT entry  
14 have the prediction values existing when their hint instruction was generated at the time of  
15 the last hit in the line in a IL1 cache entry. The restored prediction states of the BHT bits  
16 provide "hints" as to the most likely taken, or not-taken, path from a branch instruction in  
17 the line of program instructions.

18 FIGURE 2A shows a detailed embodiment of hint processor hardware logic sub-circuits  
19 which are preferably located in the same semiconductor chip having the circuits used for  
20 processing the program instructions using the hint instructions. The hint processor is  
21 shown in two parts: a "hint instruction generation" part on the right of the vertical dashed  
22 line, and a "hint instruction execution" part on the left of the vertical dashed line.

23 In the hint processor in FIGURE 2A, the "hint instruction generation" circuits have a BHT  
24 hint write register 241 into which are loaded IFAR address bits 48:58. These address bits  
25 are also received in the eleven-bit "bht index" field having locations 5-15 in a BHT hint  
26 register 242. The hint instruction operation code is internally provided into its first four

1 bit locations 0-4 comprising the “Ld\_bht\_op” field. Concurrently, all program instructions  
2 (up to 8 instructions comprising the instruction line in the selected IL1 cache entry) are  
3 copied to “branch mask creation logic” register 243, from which a “branch mask” field is  
4 formed in register 242. To form the mask, a “1” bit is stored in the branch mask field to  
5 locate each branch instruction in the line, and a “0” bit is stored in the branch mask field to  
6 locate each non-branch instruction in this field. Thus, in the branch mask field each bit  
7 positions in the mask corresponds to the position of its represented program instruction in  
8 the line. The “bht\_bits” field at bit positions 24-31 in register 242 receives the bits in the  
9 BHT field located by the current IFAR address bits 48:58.

---

10 The content of register-s 242 is outputted to a hint instruction location in the IL1 cache  
11 entry located by IFAR bits 48:58 in register 241 when a new hint instruction is required by  
12 operation 907 in the process of FIGURE 9.

13 The “hint instruction execution” circuits of the hint processor in FIGURE 2A are used by  
14 the operation 822 in the process shown in FIGURE 8. This operation restores the bits in  
15 the current BHT entry for the branch instructions in a newly refetched line of instructions.  
16 Then, the hint instruction is fetched from the L2 cache to the IL1 cache and is executed by  
17 the “hint instruction execution” circuits of the hint processor in FIGURE 2A. The  
18 execution begins when the hint instruction is transferred into hint instruction register 231  
19 in the hint processor in FIGURE 2A. Concurrently, the associated BHT entry (eight bits  
20 located by the current IFAR bits 48:58) is copied to the “curr\_bht register 232. The  
21 “branch mask” field in bits 16-23, and the “bht-bits” field in register 231 are outputted to  
22 “new BHT creation logic” circuits 238, which outputs its created BHT value to a  
23 “new\_bht” register 239, from which it is written in the BHT field located by IFAR bits  
24 48:58 to overlay the current BHT entry in the BHT. Generally, the resultant BHT is a  
25 modification of the BHT received by the “curr\_bht register 232.

26 FIGURE 2B shows the circuit logic for bit position, n, within the “new BHT creation logic”  
27 circuits 238. Bit position n is duplicated for each of the eight BHT bit positions, 0 through

1 7 comprising each BHT. Only one of the  $n$  bit positions may be changed at a time, and it is  
2 the bit position that is selected by the current IFAR address. The circuits for BHT bit  $n$   
3 comprise two logical AND gates 251 and 252 having their outputs connected to an OR  
4 circuit 254, which provides the “new\_bit ( $n$ )” output that is written into the BHT at the  
5 current IFAR selected I-index. Thus, gate 251 receives the “bht\_bits( $n$ )” bit in the  
6 “bht\_bits” field. Gate 252 receives “curr\_bht( $n$ )” bit in the “curr\_bht” field. Gate 251 is  
7 enabled by bit  $n$  in the “branch mask” field, called “branch\_mask( $n$ )”. Gates 251 and 252  
8 are alternately controlled by bit  $n$  in the “branch mask” field, wherein “branch\_mask( $n$ )”  
9 enables gate 251 and its inverted value outputted by inverter 253 disable gate 252 when  
10 gate 251 is enabled, and visa-versa. The eight bit content in the “new\_bht” register 239  
11 provides the output value written into the currently addressed BHT entry.

12 Having a L2 cache support two or more L2 lines simultaneously having copies in the IL1  
13 cache requires the L2 cache size to be at least twice as large as the IL1 cache. The L2/IL1  
14 ratio is the ratio of the number-of-L2-cache entries to the number-of-IL1 cache entries. In  
15 the detailed embodiment, the L2/IL1 ratio is a power-of-two ratio. When this ratio is two  
16 or more, it enables the L2 cache to simultaneously contain a copy of a current IL1 line, and  
17 a copy of a IL1 replacement line for the same IL1 cache entry. It is advantageous to make  
18 the L2 cache have several times the number of IL1 cache entries, in order to reduce the L2  
19 cache line thrashing caused by L2 cache misses which can delay the IL1 cache operations,  
20 when new lines of program instructions must be obtained from the system storage  
21 hierarchy. Thus at a minimum, the L2 cache should have at least twice the number of  
22 entries in the IL1 cache for a minimum ratio of two.

23 In the detailed embodiment, a L2/IL1 ratio of 32 ( $32=2^{**5}$ ) is used, which allows up to 32  
24 different L2 entries to map to each IL1 entry in the illustrated IL1 cache, which has 2048  
25 IL1 cache entries ( $2^{**11} = 2048$ ). These 11 bits are represented by bit positions 48:58 in  
26 any 64 bit address, and these bits 48:58 map into the IL1 cache the program address for a  
27 line of instructions, and the remaining high-order bits 0:47 of the 64 bit address are placed  
28 in the IL1 cache directory to identify the 64 bit address. To map any memory address into

1 the IL1 cache, the 11 bits 48:58 in the 64 bit address are used as an index into the IL1 cache  
2 to select the IL1 cache entry. The remaining high-order bits 0:47 of the 64 bit address are  
3 placed in the IL1 cache directory to identify the 64 bit address in the IL1 cache directory  
4 entry at the same index (i.e. bits 48:58) as is used to locate the IL1 cache entry.

5 The L2 cache in the detailed embodiment has 65385 L2 cache entries ( $65386 = 2^{16}$ ),  
6 whereby  $65386/2048=32$  (which is the L2/IL1 size ratio). To map any 64 bit memory  
7 address into the L2 cache, its 16 bits 43:58 are used as an index into the L2 cache to select  
8 the L2 cache entry. The remaining high-order bits 0:42 of the 64 bit address are placed in  
9 the corresponding L2 cache directory entry located therein at the same index (i.e. bits  
10 43:58) as is used to locate the associated L2 cache entry to identify the same 64 bit address  
11 in that L2 cache directory entry. Thus, any 64 bit address may be mapped into the L2  
12 cache at L2 index 43:58 having its high-order bits 0-42 placed in the corresponding L2  
13 cache directory entry at this same index 43:58; and this same 64 bit address may be  
14 mapped into the IL1 cache at IL1 index 48:58 having its high-order bits 0-47 placed in the  
15 corresponding IL1 cache directory entry at this same index 48:58

16 Using these IL1 and L2 cache sizes, the memory address of the current IL1 line (to be  
17 replaced) is identified by IFAR bits 0-47 in the current IL1 directory entry located in the  
18 IL1 cache at the IL1 index determined by bits 48:58 of the IFAR address. The current IL1  
19 line (being replaced in IL1) has a copy in a L2 cache entry located in the L2 cache located  
20 by the address identified in an "address of the first instruction" field in this IL1 directory  
21 entry. The replacing line in the L2 cache has its copy is located at IFAR index 43:58 and its  
22 L2 directory entry contains bits 0:42 of this same memory address. A hint instruction is  
23 executed during the IL1 line replacement process, as the hint instruction is fetched from  
24 the L2 cache row, to modify the BHT to provide the best available BHT predictions for the  
25 branch instructions in the newly fetched line. A new hint instruction is generated each  
26 time an instruction hit is obtained in the line to refresh the hint instruction stored in the  
27 IL1 row to insure it has the latest predictions provided in the BHT for the branch  
28 instructions in the line.

1 The hint instructions enable a program to most efficiently perform its instruction  
2 executions. The avoidance of mispredictions by this invention avoids aborting execution  
3 selections in the processor's instruction execution pipeline where the branch instruction  
4 executions are belated checked and found to be incorrect due to executing mispredicted  
5 branch instructions. Mispredictions cause much additional program delay due to  
6 additional instruction executions caused by backtracking the execution stream to correct  
7 mispredicted execution paths, requiring additional fetches of lines of instructions from the  
8 IL1 cache in a program that significantly slow the execution of the program. This invention  
9 can avoid most of the mispredicted target instruction delays, speeding up the execution of  
10 any program.

11 Detailed Description of Processes and Subprocesses used by the detailed Embodiment:

12 The process in FIGURE 8 is entered at operation 802 when program execution is started in  
13 the processor. Then operation 804 sets the processor's IFAR (instruction fetch address  
14 register) to the address of the first instruction in the program and start execution of the  
15 program. The processing performed in FIGURE 8 is concerned with hint instruction  
16 generation and use during a processor's selection and execution of program instructions in  
17 an IL1 instruction cache 201 utilizing BHT branch predictions, and using an L2 cache 212  
18 storing hint instructions during the execution of the program.

19 The next operation 806 uses the current IFAR address bit positions 48:58 as an IL1 index  
20 to locate a line of instructions in an entry in the IL1 directory 202. It is to be noted that  
21 operation 806 may be enter on the initiation of a program, and is reentered in response to  
22 an IL1 cache miss which causes operation 806 to be reentered on a loop back from the last  
23 operation 822 in FIGURE 8.

1 The next operation 807 tests the validity bit in the located IL1 directory entry. The state of  
2 the valid bit is written into a processor storage field called "valid\_IL1\_entry" which is set  
3 to a "0" state by operation 808 when the no path is taken from the operation 807 test when  
4 it indicates the IL1 directory entry is in the "invalid" state.

5 If operation 807 finds it valid, the yes path to operation 809 is taken and the  
6 "valid\_IL1\_entry" is set to the "1" state, which indicates a valid line exists in the current  
7 IL1 entry. Then operation 809 determines if the current IFAR address has a hit or miss  
8 with this valid line, and the "address of the first instruction" field is read from the IL1  
9 directory entry to determine the main memory address of the IL1 entry to be overlaid. The  
10 "address of the first instruction" field contains the high-order bits 0:47 of the memory  
11 address for locating the corresponding (associated) instruction in the IL1 cache 201 entry  
12 located by the current IFAR address bit positions 48:58. The first (or next) instruction to  
13 be executed in the program in this IL1 entry is located by bits 59 through 61 (i.e. 59:61) of  
14 the current IFAR address (used as an index in the current line of program instructions in  
15 the currently accessed IL1 cache entry).

16 An IL1 cache hit (IL1 hit) is obtained when operation 807 finds the valid bit in the valid  
17 state, and the yes path is take from operation 809 when the "address of the first  
18 instruction" field compare equal with the current IFAR bits 0:47, causing the process to go  
19 to FIGURE 9 entry B which enters operation 901 as the next operation in the process. But  
20 if operation 809 finds an unequal compare, the no path is taken to operation 810~~2~~.

21 When operation 807 finds the valid bit in the invalid state, and operation 808 sets the  
22 "valid\_IL1\_entry" field to 0, operation 810 is entered. Operation 810 accesses the L2  
23 cache directory entry located by an L2 index determined by the current IFAR bits 43:58.

24 Then, operation 812 is entered. Operation 812 tests the L2 cache entry for an L2 cache  
25 hit/miss indicated by an valid/invalid bit state in the L2 cache directory entry. If invalid,



1 the L2 cache directory does not contain a copy of the required line of program instructions  
2 for the IL1 with an accompanying hint instruction, and operation 815 is entered.

3 But if operation 812 finds a valid L2 entry, the yes path is taken to operation 813 to  
4 determine if the valid L2 entry has a L2 hit or L2 miss. An L2 miss is determined if  
5 operation 813 finds the address of the first instruction in the L2 cache directory entry  
6 mis-matches with the current IFAR bits 0-42. Then, the no path is taken to operation 814,  
7 which checks the state of the type bit in the same L2 directory entry. An L2 cache miss is  
8 then determined if operation 814 finds the D (data) type is indicated for the addressed L2  
9 cache entry, since an I (instruction) type is required for the addressed L2 cache entry if a  
10 cache hit occurs, which would allow the instructions in that line to be fetched to the IL1.  
11 However, the D type indication (L2 cache miss) requires that operation 815 be entered to  
12 use the IFAR address to fetch a line of instructions in the system main memory and store  
13 that line into the currently addressed L2 cache entry, and the corresponding L2 directory  
14 entry is validated by setting its type bit to the I state and its valid bit to the valid state.

15 Operation 815 also sets a NOP hint instruction 109 into the hint instruction field of the  
16 addressed L2 cache entry for the new L2 instruction line, which will be fetched into the IL1  
17 as a new IL1 instruction line. Then, operation 817 checks the valid state of the IL1  
18 directory entry (valid if the “valid\_IL1\_entry” field equals 1) to determine if the  
19 corresponding IL1 entry contains a valid IL1 cache line which is about to be replaced in  
20 the IL1 entry.

21 When operation 817 finds the “valid\_IL1\_entry” set to the “0” (indicating a invalid state  
22 for the IL1 entry), there is no IL1 line to be overlaid. Therefore the IL1 entry is in a  
23 condition to receive the new replacing instruction line from the L2 cache, since there is no  
24 current IL1 entry to replace, and the no path is taken to operation 822.

25 Then, operation 822 accesses the L2 cache row addressed by IFAR bits 43:58 and transfers  
26 it to the currently accessed IL1 entry; that row contains an instruction line having “eight

1 program instructions”, and a hint instruction. This hint instruction is also forwarded to  
2 hint instruction register 231 in the hint instruction processor 206 shown in detail in  
3 FIGURE 2A, which then executes the hint instruction newly written into the accessed IL1  
4 entry from the L2 cache entry. Also, the current BHT entry is replaced with a modified  
5 BHT entry generated in the hint processor 206, as explained herein for FIGURES 2A, 2B  
6 and 13.

7 However, if operation 817 finds the “valid IL1\_entry” set to the “1” (indicating a valid IL1  
8 entry will be replaced which does not match the current IFAR bits), the process then  
9 follows its yes path to operation 816 which assigns a “IL1\_hint\_wr\_addr” field in a  
10 predetermined storage location and stores in it the IL1 cache index of the hint instruction  
11 which is provided by current IFAR bits 48:58. Operation 817 also assigns a  
12 “IL2\_hint\_wr\_addr” field in another predetermined storage location to the copy of the line  
13 about to be replaced in the IL1 cache, and stores its L2 cache index, which is the  
14 concatenation of bits 43:47 in the “address of the first instruction” field of the IL1  
15 directory entry located by IFAR bits 48:58 (now stored in the “IL1\_hint\_wr\_addr” field).  
16 Then operation 816 accesses the L2 directory entry at the address stored in the  
17 “IL2\_hint\_wr\_addr” field, and goes to operation 818.

18 For finding the L2 line address of the line to be fetched, operation 816 determines the L2  
19 address for the current line in IL1 by assigning a “IL1\_hint\_wr\_addr” field in a  
20 predetermined storage location to receive the current entry’s IL1 index, which is set to  
21 IFAR bits 48:58.

22 For locating the L2 copy of the current IL1 entry about to be replaced (which locates where  
23 the castout hint instruction is to be stored in the L2 cache), operation 816 assigns an  
24 “IL2\_hint\_wr\_addr” field in another predetermined storage location, and this field  
25 receives an L2 cache index equal to the concatenation of bits 43:47 of the “Address of the  
26 first instruction” field of the IL1 directory entry located by IFAR bits 48:58 in the

1    **“IL1\_hint\_wr\_addr” field. Then operation 816 accesses the L2 directory entry at the**  
2    **address indicated in the “IL2\_hint\_wr\_addr” field, and goes to operation 818.**

3    **Operation 818 tests if this L2 entry is valid and if it contains a copy of the required IL1 line**  
4    **by comparing the “address of the first instruction” field in the L2 directory and the**  
5    **“address of the first instruction” field in the current IL1 directory entry. Furthermore,**  
6    **operation 818 also checks the “type” field in this L2 directory entry for the “I” state. If all**  
7    **of these tests by operation 818 are successful, the instruction line being replaced in IL1 has**  
8    **a copy in the L2 cache, and the process takes the yes path to operation 820. Operation 820**  
9    **writes the hint instruction from the current entry in the IL1 cache (indexed in IL1 by the**  
10    **current IFAR bits 48:58) to the hint instruction field of the L2 cache entry (in the row**  
11    **located in the L2 cache by the current content of the IL2\_hint\_wr\_addr field).**

12    **However, if operation 818 is unsuccessful, there is no valid instruction line to be replaced in**  
13    **IL1 and it cannot have a copy in the L2 cache, and the process goes to operation 822.**  
14    **Operation 822 loads the currently addressed IL1 row from the currently accessed L2 cache**  
15    **entry by transferring the “eight program instructions” field and the hint instruction field**  
16    **from the L2 cache entry located by IFAR bits 43:58. This hint instruction is also**  
17    **forwarded to the hint instruction processor in FIGURE 2A, which then executes the hint**  
18    **instruction process shown in FIGURE 13, and the FIGURE 13 process operates in parallel**  
19    **with a continuation of the process in FIGURE 8.**

20    **The process in FIGURE 13 is entered at operation 1301 for testing during the current**  
21    **IFAR cycle if the received instruction is a hint instruction. If the test does not find a hint**  
22    **instruction, the process takes the no path to its exit. If a hint instruction is found by**  
23    **operation 1301, the process goes to operation 1302 to test if the hint instruction operation**  
24    **code is the ld\_bht\_op field, or a NOP field. If a NOP is found, the process goes from**  
25    **operation 1301 to the exit in FIGURE 13. If a ld\_bht\_op field is found by operation 1302,**  
26    **the BHT write update path is followed (it uses the triggers “wr\_en hold 1” 236 and “wr\_en**  
27    **hold 2” 237 in FIGURE 2A) to send an a hint instruction interpretation enable signal.**

1 Then the next operation 1303 is performed, and it reads the BHT entry indexed by the  
2 bht\_index field in the current hint instruction, and copies it into the curr\_bht register 232.

3 Then, operation 1304 (using the hint instruction in register 231) generates a new BHT  
4 entry for being set in a "new\_bht" register. It uses logical AND/OR bit by bit functions as  
5 previously explained herein for FIGURE 2B, in which each of the respective bit n is  
6 generated for the "new\_bht" register as: (the nth curr\_bht bit AND the inversion of the nth  
7 "branch\_mask" bit) OR (the nth bht\_bits bit AND the nth "branch\_mask" field in the  
8 hint instruction).

---

9  
10 Finally, operation 1305 stores the eight bit "new\_bht" field value in the BHT entry  
11 currently indexed by the content of the "bht\_index" field of the hint instruction. The  
12 process in FIGURE 13 then exits and goes to FIGURE 8 operation 806 to again read the  
13 IL1 directory entry indexed by IFAR bits 48:58. Then operation 807 again tests this same  
14 IL1 directory entry for validity; and since it has been made valid, the next operation 809  
15 sets the "valid\_IL1\_entry" to 1, and finds that now the current IFAR bits 0:47 matches the  
16 "address of the first instruction" field in the new content in the same IL1 directory entry.  
17 An IL1 hit then occurs and the process goes to FIGURE 9 entry point B.

18 Operation 901 is entered in Figure 9 at entry point B. At operation 901, the IL1 cache line  
19 is fetched into the "Eight Program Instructions" register 309, and the associated hint  
20 instruction into the "Hint Instructions" register 231. Next, the BHT entry indexed by the  
21 IFAR bits 48:58 is accessed, and its BHT prediction bits are fetched into the "Eight  
22 Predictions" register 308.

23 Then operation 903 uses the IFAR bits 59:61 to locate a "first instruction" in the "Eight  
24 Program Instructions" register 309 (Instructions before the "first instruction", if any, will  
25 be ignored).

1 The next operation 904 is tests if there is any branch instruction in the "Eight Program  
2 Instructions" register 309 at or after the "first instruction"? If "no", operation 906 is  
3 entered and designates a "fetch group" as the instructions from the "first instruction" to  
4 the end of register 309. Then, a "Predicted\_IFAR" field in logic 311 is set to the address of  
5 the next sequential instruction after the "fetch group", and the process goes to operation  
6 926.

7 But if operation 904 takes its "yes" path, the process performs operation 907, which  
8 generates a new hint instruction in the currently selected IL1 cache row. This is done by  
9 the hint processor 206 (in FIGURE 2A) filling its BHT Hint register 242 with the following:  
10 bits 0:4 with "ld\_bht\_op", bits 5:15 with IFAR bits 48:58, bits 16:23 with an 8-bit "branch  
11 mask" field containing a 1 in the positions where there is a branch and 0 in other positions,  
12 bits 24:31 with the 8-bit BHT prediction. Then the hint processor stores IFAR bits 48:58 in  
13 the BHT Hint Write Entry register 241, and operation 907 finally stores the content of the  
14 BHT Hint register in the IL1 Hint Instruction Cache entry indexed by BHT Hint Write  
15 Entry register 241.

16 Then the next operation 911 determines if any branch bit in the "Eight Predictions"  
17 register 308 (which in FIGURE 3 receives the last-outputted BHT field) indicates an  
18 unconditional branch predicted taken, or a conditional branch predicted taken? If the  
19 "yes" path is determined, operation 912 is entered and logic 311 in FIGURE 3 sets  
20 "Predicted\_IFAR" address to the target of the first of these branches and designates this  
21 branch as the "last instruction", and operation 921 is entered.

22 .

23 But if the "no" path is determined by operation 911, then operation 914 is entered and logic  
24 311 in FIGURE 3 sets "Predicted\_IFAR" address to the instruction next sequential to the  
25 last instruction fetched: and the last instruction in the Eight Instructions" register 309 is  
26 designated as the "last instruction", and operation 921 is entered.

1 Operation 921 then forms the "fetch group" to contain all instructions between the "first  
2 instruction" and the "last instruction" determined in the Eight Program Instructions  
3 register 309. For each branch instruction in the "fetch group", operation 926 obtains an  
4 invalid entry in the Branch Information Queue (BIQ) 313 in FIGURE 3, and FIGURE 4  
5 shows BIQ 313 in more detail. Then in BIQ 313, operation 921 sets the valid bit to 1 state  
6 in this BIQ entry, loads the address of the branch into an "Address of the branch" field  
7 401, loads the branch target address in the "Predicted address" field 402 if the branch is  
8 predicted taken or loads the next sequential address in the "Predicted address" field 402 if  
9 the branch is predicted not-taken, and stores the n-th bit in the "Eight Predictions"  
10 register 308 in a "BHT bit" field 403 if the branch is at position "n" in the fetch group.  
11 Finally, operation 921 places the branch instruction in Branch Issue Queue 216 314 for its  
12 subsequent execution. Then the process goes to operation 926.

13 Operation 926 forwards the "fetch group" to Instruction Decode Unit (IDU) 208 shown in  
14 FIGURES 2 and 3 and performs the Instruction Decode and Dispatch process shown in  
15 FIGURE 11 (this is also described in previously-cited US Patent 6598152, granted July 22,  
16 2003 filed application docket number POU919990174 having USPTO serial number  
17 09/436264). The process in FIGURE 11 may precede in parallel with the process in  
18 FIGURE 9. When the process in FIGURE 9 is completed, the process goes to entry point  
19 C in FIGURE 10.

20 When the process in FIGURE 11 is entered, operation 1101 is performed to determine if a  
21 "fetch group" was forwarded by the instruction fetch unit (IFU) and if it is the "fetch  
22 group" identified in the current IFAR cycle (i.e. addressed by the current IFAR setting). If  
23 the test by operation 1101 finds no "fetch group" has been forwarded for the current IFAR  
24 cycle, the "no" path is taken to the exit the process in FIGURE 11.

25 However if the test by operation 1101 finds the "fetch group" is for the current IFAR cycle,  
26 the "yes" path is taken to operation 1102, which is performed by IDU 208, which then  
27 forms one or more "dispatch groups" from the received "fetch group" following the rules

1 of dispatch group formation. (These rules are: Not more than five instructions per group,  
2 At most one branch instruction in each dispatch group, and The fifth slot in the dispatch  
3 group is reserved for branch instructions only and if there is not enough instructions to fill  
4 all the slots in the dispatch group which have inserted NOPs).

5 Then operation 1103 obtains an invalid entry in the Global Completion Table (GCT) ~~244~~ in  
6 209 shown in FIGURE 2 and fill its fields with the information for the dispatch group and  
7 validates the entry.

8 Finally, operation 1103 places each of the instructions in the “dispatch group” in the issue  
9 queue, and makes it available to the process shown in FIGURE 12 for operation 926.

10 The FIGURE 12 process is done by the Branch Issue Queue 314 and Branch Execution  
11 Logic ~~346~~ unit 217A with outputs 316A and 316B shown in FIGURE 3. In FIGURE 12  
12 the process performs Instruction issue and instruction execution operations, in which  
13 operation 1201 is entered. Operation 1201 determines if there is any valid Instruction in  
14 the Issue Queue for which all the operands are known? If “no”, the process waits one cycle  
15 and then again performs operation 1201 until a valid instruction is detected in the Issue  
16 Queue for which all operands are known.

17 Operation 1203 is entered from the “yes” path from operation 1201. Then, operation 1203  
18 forwards the detected Instruction to its proper execution unit 217A - 217D, which is one of  
19 the execution units shown in the Instruction Execution Units (IEU) 217 in FIGURE 2,  
20 which involves sending a branch instruction to the branch execution unit 217A, a load/store  
21 instruction to the load/store execution unit 217D, a fixed-point instruction to the fixed-point  
22 execution unit 217B, and a floating-point instruction to the floating-point execution unit  
23 217C. When the respective execution unit receives an instruction, it executes the  
24 instruction.

1   Operation 1203 forwards the instruction to its proper execution unit in the instruction  
2   execution unit 217 in FIGURE 2, and then operation 1204 executes the instruction. The  
3   process in FIGURE 12 then goes back to operation 1201 to repeat its operations for another  
4   valid instruction in the issue queue.

5   When operation 1203 forwards a conditional branch instruction to the branch execution  
6   logic 217A, it determines if the actual “branch taken/not taken” path is the same as the  
7   predicted “branch taken/not taken” path made by the BHT bit prediction for this  
8   instruction. If the actual and predicted are the same, the process in FIGURE 10 continues  
9   the predicted instruction stream. But if the determination finds they are not the same, then  
10   the target instruction selected in the predicted instruction stream is in error, and the  
11   execution results of that branch target execution, and of all of its following instruction  
12   executions, must be flushed (eliminated) from the execution results for the current  
13   program, and they must be replaced by executing the instructions beginning with the  
14   actual target instruction determined by the actual execution of the wrongly predicted  
15   branch instruction.

16   In FIGURE 10, operation 1001 determines if the current instruction is being executed in  
17   the current cycle is a branch instruction. If no branch instruction is being executed, the  
18   program execution sequence is not affected; then the “no” path is taken to operation 1002,  
19   which occurs for most instructions in a program.. But if the currently executing  
20   instruction is a branch instruction, the “yes” path is taken to operation 1003.

21   When the “no” path is taken from operation 1001 to operation 1002, operation 1002  
22   determines if any non-branch flush signal has been received. Mostly non-flush signals are  
23   not received because the predictions are correct, and the “no” path is taken to operation  
24   1002\_5 which sets the IFAR to the “predicted\_IFAR” address value. Then the subprocess  
25   in FIGURE 10 is ended, and the process goes to FIGURE 8 entry point A.



1 However, if the “yes” path is taken from operation 1005 to operation 1006, operation 1006  
2 sets IFAR to the non-branch flush address received. Then the subprocess in FIGURE 10 is  
3 ended, and the process goes to FIGURE 8 entry point A.

4 When a branch instruction is being executed, operation 1003 is performed using the  
5 Branch Information Queue (BIQ) hardware in FIGURE 4, and the operation reads the  
6 current BHT bit 403 and the Predicted Address 402 (for predicting the outcome of the  
7 currently executed branch instruction) in the current BIQ entry in BIQ 313. Then,  
8 operation 1003 determines if the branch instruction is mispredicted by finding if the valid  
9 bit 404 indicates the invalid state, or the actual target address is different from the  
10 predicted address 402. That is, the predicted and actual addresses are compared, and if  
11 they do not have the same value, this branch instruction has a misprediction; then  
12 operation 1003 takes its “yes” path to operation 1007.

13 The usual case for operation 1003 is to find no misprediction (i.e. the compared predicted  
14 and actual addresses have the same value), and then the “no” path is taken to operation  
15 1004. Operation 1004 sets IFAR to the “Predicted IFAR” value, which is the address of the  
16 target instruction of this executed branch instruction. Then operation 1011 is entered, and  
17 the BIQ entry is released for this executed branch instruction by setting its BIQ valid bit  
18 404 to “0” state. The subprocess in FIGURE 10 is ended, and it goes to FIGURE 8 entry  
19 point A.

20 However, when the “yes” path from operation 1003 to 1007 is taken, a determination is  
21 made if the prediction by BHT bit 403 is correct. It is possible for the state of BHT bit 403  
22 to be correct and for a misprediction to nevertheless exist. If operation 1007 finds the BHT  
23 bit prediction is not correct, operation 1012 is entered. But if operation 1007 finds the BHT  
24 bit prediction is correct, operation 1017 is entered.

25 If the BHT bit prediction is correct, and operation 1017 is entered, then operation 1017 sets  
26 “Execution IFAR” to the target address of the branch instruction, and sets IFAR to the

1   **“Execution IFAR” value, and flushes all instructions from the instruction pipeline**  
2   **following the current branch instruction. Finally, operation 1021 releases the BIQ entry**  
3   **for the executed branch instruction by setting its valid bit to the “0” state. The process**  
4   **then goes to FIGURE 8 entry point A.**

5   **But if operation 1007 finds the BHT bit prediction is not correct, operation 1012 is entered**  
6   **to determine if the branch outcome is “taken”. If “taken”, operation 1014 sets “the**  
7   **“Execution IFAR” value to the target address of the branch instruction. If “not taken”,**  
8   **operation 1016 sets the “Execution IFAR” value to the value obtained by adding 4 to the**  
9   **“Address of the branch” field in the BIQ entry for the executed branch to generate the**  
10   **address of the next sequential instruction in the program..**

11   **When performed, each operation 1014 or 1016 enters operation 1018, which sets IFAR to**  
12   **the “Execution IFAR” value, and the execution results obtained for all instructions**  
13   **following the branch are flushed from instruction pipeline.**

14   **Then, operation 1019 sets a BHT\_write\_addr register 318 to the “address of the branch”**  
15   **field obtained from the BIQ entry for the executed branch. The BHT\_write\_data is set to**  
16   **1, if the branch outcome is “taken”, else it is set to 0, and this value is written over the**  
17   **current BHT bit in the BHT to insure that it is corrected.**

18   **The next operation 1021 is then performed, and it releases the BIQ entry for the executed**  
19   **branch instruction by setting its valid bit to the “0” state. The process then goes to**  
20   **FIGURE 8 entry point A to repeat its operation which has been previously described**  
21   **herein..**

22   **While I have described the preferred embodiment of my invention, it will be understood**  
23   **that those skilled in the art, both now and in the future, may make various improvements**  
24   **and enhancements which fall within the scope of the claims, which follow. These claims**

1 should be construed to maintain the proper protection for the invention first disclosed  
2 here.

3 The invention claimed is:

4 Claim 15. (Currently amended) A branch prediction process for a computer  
5 system for improving branch prediction rate when using a branch history table, as  
6 ~~defined by claim 14, further~~ comprising:

7 determining if a program instruction processor (processor) has an access hit (hit) or access  
8 miss (miss) in an instruction cache (I-cache) when utilizing an instruction address (IFAR  
9 address) in attempting to select a program instruction for execution by the processor,

10 generating a hint instruction when the program instruction is a branch in response to a hit  
11 occurring during the determining operation, storing the hint instruction in association with  
12 a copy of an instruction line containing the program instruction in a storage hierarchy of  
13 the computer system, the hint instruction storing BHT prediction fields obtained from a  
14 copy of a current BHT entry associated with the instruction line when the hit occurs, and  
15 storing a branch mask in the hint instruction for locating an associated BHT field  
16 indicating the BHT field associated with the location of the program instruction in the  
17 instruction line, and

18 transferring the copy of the instruction line and associated hint instruction from the  
19 storage hierarchy to the I-cache in response to a miss occurring during the determining  
20 operation, and executing the hint instruction to restore a BHT prediction field in a current  
21 BHT entry to the state of a BHT field in the hint instruction located by the branch mask,  
22 and

23 the generating operation of generating hint instructions being performed by a hint  
24 processor operating in parallel with the program instruction processor, and

1    executing a hint instruction when the hint instruction is received in the I-cache by testing  
2    an operation code field in the hint instruction to determine if a completed hint instruction  
3    is indicated or if a no-operation state is indicated for the hint instruction, and continuing  
4    the executing process only if a completed hint instruction is indicated by performing the  
5    following operations:

6    reading a BHT entry in the BHT located at an index determined by a bht\_index field in the  
7    hint instruction, and storing the BHT entry in a curr\_bht register,

8    logically ANDing an Nth bit in an inversion of the branch\_mask field in the hint instruction  
9    with an Nth bit in the curr\_bht register, where N is the bit position of the current  
10    instruction in the instruction line, and logically ANDing the Nth bit in a branch\_mask field  
11    with an Nth bit in a bht\_bits in the hint instruction,

12    logically ORing outputs of the two logical ANDing operations to provide an Nth bit output,  
13    and setting an Nth bit in a new\_bht register to the Nth bit output,

14    receiving without change in the new\_bht register at bit locations other than at the Nth bit  
15    location the bits in the curr\_bht register at corresponding bit locations other than the Nth  
16    location, and

17    setting the content of the new\_bht register into the current BHT entry in the BHT to  
18    restore the BHT entry to its last prediction state for the current instruction.

19    16. (Currently Amended) The ~~A~~ branch prediction process for a computer system for  
20    improving branch prediction rates when using a branch history table as defined by claim  
21    15, further comprising:

22    performing all of the hint instruction operations in a hint instruction processor.

- 1 17. (Currently Amended) The ~~A~~ branch prediction process for a computer system for
  - 2 improving branch prediction rates when using a branch history table as defined by claim
  - 3 16, further comprising:
  - 4 performing all hint instruction operations and all program instruction processor
  - 5 operations in a single semiconductor chip.
-

1       **ABSTRACT OF THE DISCLOSURE**

2   **Apparatus and methods implemented in a processor semiconductor logic chip for**  
3   **providing novel "hint instructions" that uniquely preserve and reuse branch predictions**  
4   **replaced in a branch history table (BHT). A branch prediction is lost in the BHT after its**  
5   **associated instruction is replaced in an instruction cache. The unique "hint instructions"**  
6   **are generated and stored in a unique instruction cache which associates each hint**  
7   **instruction with a line of instructions. The hint instructions contains the latest branch**  
8   **history for all branch instructions executed in an associated line of instructions, and they**  
9   **are stored in the instruction cache during instruction cache hits in the associated line.**  
10   **During an instruction cache miss in an instruction line, the associated hint instruction is**  
11   **stored in a second level cache with a copy of the associated instruction line being replaced**  
12   **in the instruction cache. In the second level cache, the copy of the line is located through**  
13   **the instruction cache directory entry associated with the line being replaced in the**  
14   **instruction cache. Later, the hint instruction can be retrieved into the instruction cache**  
15   **when its associated instruction line is fetched from the second level cache, and then its**  
16   **associated hint instruction is also retrieved and used to restore the latest branch**  
17   **predictions for that instruction line. In the prior art this branch prediction would have**  
18   **been lost. It is estimated that this invention improves program performance for each**  
19   **replaced branch prediction by about 80%, due to increasing the probability of BHT bits**  
20   **correctly predicting the branch paths in the program from about 50% to over 90%. Each**  
21   **incorrect BHT branch prediction may result in the loss of many execution cycles, resulting**

- 1 in additional instruction re-execution overhead when incorrect branch paths are belatedly
- 2 discovered.